

# BANGALORE INSTITUTE OF TECHNOLOGY

K.R ROAD, V.V PURAM, BANGALORE-560 004

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING



## VISUAL PROGRAMMING AND DATABASE APPLICATIONS LABORATORY MANUAL

(As per VI Semester V.T.U Syllabus of 2004-2008)

**Prepared By:**

- Mrs. T. Asha. M.E  
Asst Professor
- Mrs. H. Roopa. M.Tech  
Lecturer

**VISUAL PROGRAMMING AND DATABASE LABORATORY**  
**(Common to CSE/ISE)**

Sub Code: CSL67  
 Hrs/Week: 03  
 Total Hrs: 42

IA Marks: 25  
 Exam Hours: 03  
 Exam Marks: 50

1. Consider the Insurance Database given below. The primary keys are underlined and the datatypes are specified.

**PERSON** (Driver-id:string, name:string, address:string)

**CAR** (Regno:string, model:string, year:int)

**ACCIDENT** (Report-number:int, date:date, location:string)

**OWNS** (Driver-id:string, regno:string)

**PARTICIPATED** (Driver-id:string, regno:string, report-no:int, damageamount:int)

- i) Create the above tables by properly specifying the primary keys and the foreign keys.
- ii) Enter at least five tuples for each relation.
- iii) Demonstrate how you
  - a. Update the damage amount for the car with a specific **Regno** in the accident with report number 12 to 25000.
  - b. Add a new accident to the database.
- iv) Find the total number of people who owned cars that were involved in accidents in 2002.
- v) Find the number of accidents in which cars belonging to a specific model were involved.
- vi) Generation of suitable reports.
- vii) Create a suitable front end for querying and displaying the results. **(Page-5)**

2. Consider the following relations for an order processing database application in a company.

**CUSTOMER** (Cust #: int, Cname: string, City: string)

**ORDER** (Order #: int, Odate: date, Cust #: int, Ord-Amt: int)

**ORDER-ITEM** (Order #: int, Item #: int, qty: int)

**ITEM** (Item #: int, Unit Price: int)

**SHIPMENT** (Order #: int, Warehouse #: int, Ship-Date: date)

**WAREHOUSE** (Warehouse #: int, City: string)

- i) Create the above tables by properly specifying the primary keys and the foreign keys.
- ii) Enter at least five tuples for each relation.

- iii) Produce a listing: **CUSTNAME, NO\_OF\_ORDERS, AVG\_ORDER\_AMT**, where the middle column is the total number of orders by the customer and the last column is the average order amount for that customer.
- iv) List the **Order#** for the orders that were shipped from all the warehouses that the company has in a specific city.
- v) Demonstrate how you delete **Item#** 10 from the **ITEM** table and make that field *null* in the **ORDER-ITEM** table.
- vi) Generation of suitable reports.
- vii) Create a suitable front end for querying and displaying the results. **(Page-14)**

3. Consider the following database of student enrollment in course and books adopted for each course .

**STUDENT** (regno:string, name:string, major:string, bdate:date)

**COURSE** (course#:int, cname:string, dept:string)

**ENROLL** (regno:string, course#:int, sem:int, marks:int )

**BOOK\_ADOPTION** (course#:int., sem:int, book-ISBN:int)

**TEXT**(book-ISBN:int, book-title:string, publisher:string, authour:string).

- i) Create the above tables by properly specifying the primary keys and the foreign keys .
- ii) Enter atleast five tuples for each relation.
- iii) Demonstrate how you add a new text book to the database and make this book be adopted by some department.
- iv) Produce a list of text books (include Course #,Book-ISBN, Book-title) in the alphabetical order for courses offered by the 'CS' department that use more than two books.
- v) List any department that has all its adopted books published by a specific publisher.
- vi) Generation of suitable reports.
- vii) Create suitable front end for quering and displaying the results. **(Page-26)**

4. Consider the following relations for the details maintained by a book dealer.

**AUTHOR** (Author-id: int, Name: string, City: string, Country: string)

**PUBLISHER** (Publisher-id: int, Name: string, City: string, Country: string)

**CATALOG** (Book-id: int, title: string, author-id: int, Publisher-id: int, Category-id: int, Year: int, Price: int)

**CATEGORY** (Category-id: int, Description: string)

**ORDER-DETAILS** (Order-no : int, Book-id: int, Quantity: int)

- i) Create the above tables by properly specifying the primary keys and the foreign keys.
- ii) Enter at least five tuples for each relation.

- iii) Give the details of the authors who have 2 or more books in the catalog and the price of the books is greater than the average price of the books in the catalog and the year of publication is after 2000.
- iv) Find the author of the book which has maximum sales.
- v) Demonstrate how you increase the price of books published by a specific publisher by 10%.
- vi) Generation of suitable reports.
- vii) Create a suitable front end for querying and displaying the results **(Page-36)**

5. Consider the following database for a banking enterprise

**BRANCH** (branch\_name: string, branch\_city: string, assets: real)

**ACCOUNT** (accno: int, branch\_name: string, balance: real)

**CUSTOMER** (customer\_name: string, customer\_street: string, city: string)

**DEPOSITOR** (customer\_name: string, accno: int)

**LOAN** (loan\_number: int, branch\_name: string, amount: real)

**BORROWER** (customer\_name: string, loan\_number: int)

- i) Create the above tables by properly specifying the primary keys and the foreign keys.
- ii) Enter atleast five tuples for each relation.
- iii) Find all the customers who atleast two accounts at the **MAIN** branch.
- iv) Find all the customers who have an account at **all** branches located in a specific city.
- v) Demonstrate how you delete all account tuples at every branch located in a specific city.
- vi) Generation of suitable reports.
- vii) Create suitable front end for querying and displaying the results. **(Page-50)**

# **PROBLEM 1**

## **PROBLEM STATEMENT:**

Consider the Insurance Database given below. The primary keys are underlined and the datatypes are specified.

PERSON (Driver-id:string, name:string, address:string)

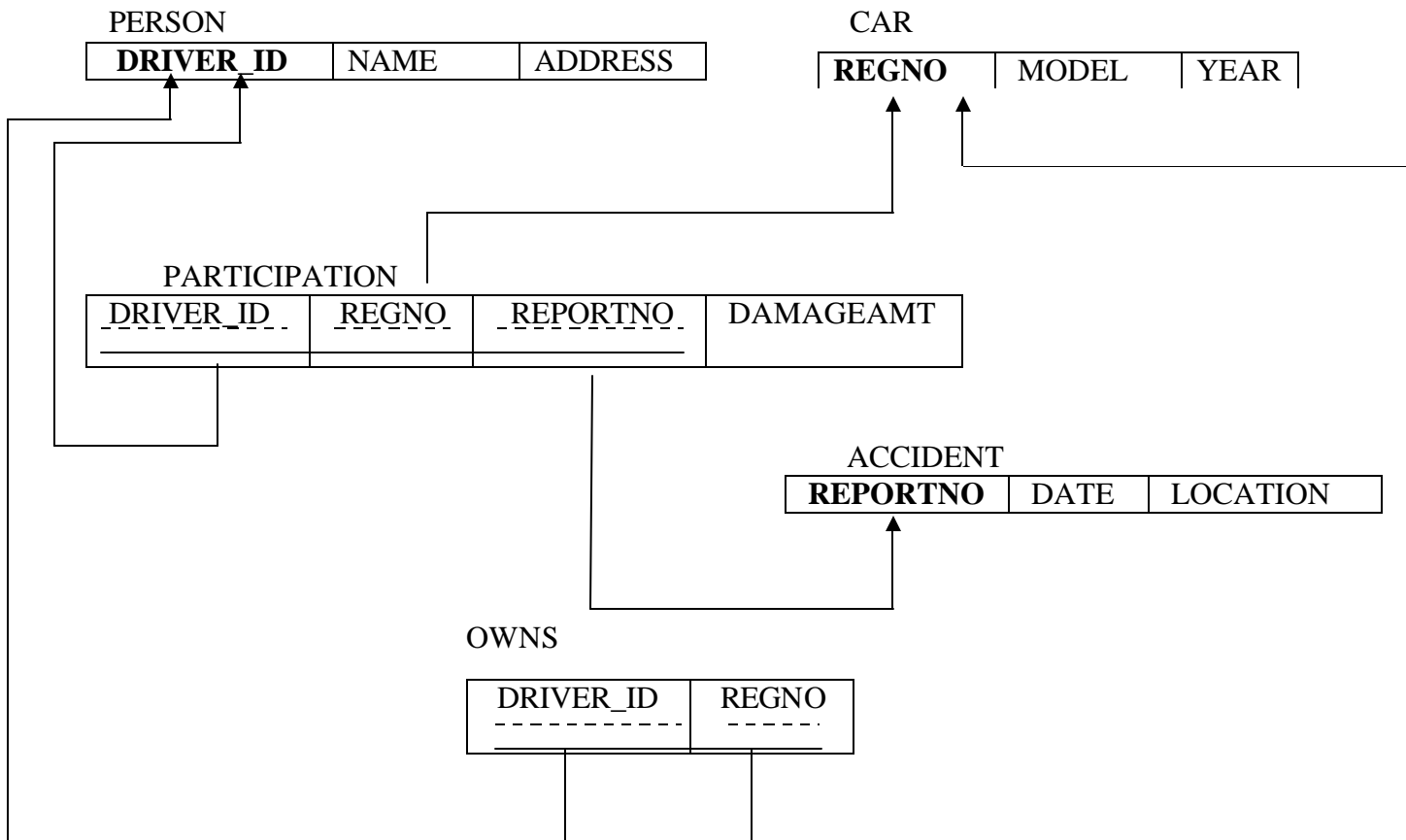
CAR (Regno:string, model:string, year:int)

ACCIDENT (Report-number:int, date:date, location:string)

OWNS (Driver-id:string, regno:string)

PARTICIPATED (Driver-id:string, regno:string, report-no:int, damageamount:int)

- i) Create the above tables by properly specifying the primary keys and the foreign keys.
- ii) Enter at least five tuples for each relation.
- iii) Demonstrate how you
  - a. Update the damage amount for the car with a specific **Regno** in the accident with report number 12 to 25000.
  - b. Add a new accident to the database.
- iv) Find the total number of people who owned cars that were involved in accidents in 2002.
- v) Find the number of accidents in which cars belonging to a specific model were involved.
- vi) Generation of suitable reports.
- vii) Create a suitable front end for querying and displaying the results.

**SCHEMA DESCRIPTION:**

- ▶ The table PERSON contains the attributes namely driver\_id, name & address where the primary key is driver\_id.
- ▶ The table CAR contains the attributes such as model, year & register number where the primary key is register number.
- ▶ The table ACCIDENT contains the attributes such as report number, date & location where the primary key is report number.
- ▶ The table OWNS contains the attributes driver id & regno, both the attributes are foreign key & they together constitute the composite key.

- ▶ The table PARTICIPATED contains the attributes driver id, regno, report number & damage amount. Here driver id , regno , report number are foreign keys and they constitute the composite key

**TABLE CREATION:**

**CREATE TABLE PERSON**

```
(
  DRIVER_ID VARCHAR (10) PRIMARY KEY,
  NAME VARCHAR (10) NOT NULL,
  ADDRESS VARCHAR (20)
);
```

**CREATE TABLE CAR**

```
(
  REG_NO VARCHAR (10) PRIMARY KEY,
  MODEL VARCHAR (10) NOT NULL,
  YEAR NUMBER (4)
);
```

**CREATE TABLE ACCIDENT**

```
(
  REPORT_NO NUMBER (10) PRIMARY KEY,
  DDATE DATE,
  LOCATION VARCHAR (10) NOT NULL
);
```

**CREATE TABLE OWNS**

```
(
  DRIVER_ID VARCHAR (10) REFERENCES PERSON (DRIVER_ID),
  REG_NO VARCHAR (10) REFERENCES CAR (REG_NO)
);
```

**CREATE TABLE PARTICIPATED**

```
(
  DRIVER_ID VARCHAR (10) REFERENCES PERSON (DRIVER_ID),
  REG_NO VARCHAR (10) REFERENCES CAR (REG_NO),
  REPORT_NO NUMBER (10) REFERENCES ACCIDENT (REPORT_NO),
  DAMAGE_AMOUNT NUMBER (10)
);
```

**VALUE INSERTION:**

**INSERT INTO PERSON VALUES  
(&DRIVER\_ID,'&NAME','&ADDRESS');**

**Enter value for driverid: 001**

**Enter value for name: ADARSH**

**Enter value for address: JPNAGAR**

**old 1: insert into person values('&driverid','&name','&address')**

**new 1: insert into person values('001','ADARSH','JPNAGAR')**

**Enter value for driverid: 008**

**Enter value for name: ANOOP**

**Enter value for address: JAYANAGAR**

**old 1: insert into person values ('&driverid','&name','&address')**

**new 1: insert into person values ('008','ANOOP','JAYANAGAR')**

**Enter value for driverid: 022**

**Enter value for name: KIRAN**

**Enter value for address: JAYANAGAR**

**old 1: insert into person values('&driverid','&name','&address')**

**new 1: insert into person values('022','KIRAN','JAYANAGAR')**

**Enter value for driverid: 003**

**Enter value for name: ABHISHEK**

**Enter value for address: PEENYA**

**old 1: insert into person values('&driverid','&name','&address')**

**new 1: insert into person values('003','ABHISHEK','PEENYA')**

**INSERT INTO CAR VALUES**

**(&REG\_NO','&MODEL','&YEAR');**

**Enter value for regno: 001**

**Enter value for model: SKODA**

**Enter value for year: 2002**

**old 1: insert into car values('&regno','&model','&year')**

**new 1: insert into car values('001','SKODA','2002')**

**Enter value for regno: 008**  
**Enter value for model: BMW**  
**Enter value for year: 2002**

**old 1: insert into car values('&regno','&model','&year')**  
**new 1: insert into car values('008','BMW','2002')**

**Enter value for regno: 003**  
**Enter value for model: MARUTI**  
**Enter value for year: 2003**  
**old 1: insert into car values('&regno','&model','&year')**  
**new 1: insert into car values('003','MARUTI','2003')**

**Enter value for regno: 022**  
**Enter value for model: HYUNDAI**  
**Enter value for year: 2001**  
**old 1: insert into car values('&regno','&model','&year')**  
**new 1: insert into car values('022','HYUNDAI','2001')**

**INSERT INTO ACCIDENT VALUES**  
**('&REPORT\_NO','&DATE','&LOCATION');**

**Enter value for report no: 1**  
**Enter value for date: 1-JAN-2002**  
**Enter value for location: RJNGR**  
**old 1: insert into accident values('&reprotno','&date','&location')**  
**new 1: insert into accident values('1','1-JAN-2002','RJNGR')**

**Enter value for report no: 2**  
**Enter value for date: 3-FEB-2003**  
**Enter value for location: JNGR**  
**old 1: insert into accident values('&reprotno','&date','&location')**  
**new 1: insert into accident values('2','3-FEB-2003','JNGR')**

**Enter value for reportno: 3**  
**Enter value for date: 5-APR-2003**  
**Enter value for location: PEENYA**  
**old 1: insert into accident values('&reprotno','&date','&location')**  
**new 1: insert into accident values('3','5-APR-2003','PEENYA')**

**Enter value for reprotno: 201**  
**Enter value for date: 2-mar-2003**

**Enter value for location: JPNGR**

**old 1: insert into accident values('&reprotno','&date','&location')**

**new 1: insert into accident values('201','2-mar-2003','JPNGR')**

**INSERT INTO OWNS VALUES**

**('&DRIVER\_ID','&REG\_NO');**

**Enter value for driverid: 001**

**Enter value for regno: 001**

**old 1: insert into owns values('&driverid','&regno')**

**new 1: insert into owns values('001','001')**

**Enter value for driverid: 008**

**Enter value for regno: 008**

**old 1: insert into owns values('&driverid','&regno')**

**new 1: insert into owns values('008','008')**

**Enter value for driverid: 022**

**Enter value for regno: 022**

**old 1: insert into owns values('&driverid','&regno')**

**new 1: insert into owns values('022','022')**

**Enter value for driverid: 003**

**Enter value for regno: 003**

**old 1: insert into owns values('&driverid','&regno')**

**new 1: insert into owns values('003','003')**

**INSERT INTO PARTICIPATED VALUES**

**('&DRIVER\_ID','&REGNO','&REPORT\_NO','&DAMAGE\_AMOUNT');**

**Enter value for driverid: 001**

**Enter value for regno: 001**

**Enter value for reprotno: 1**

**Enter value for damageamount: 9000**

**old 1: insert into participated values('&driverid','&regno','&reprotno','&damageamount')**

**new 1: insert into participated values('001','001','1','9000')**

**Enter value for driverid: 022**

**Enter value for regno: 022**

**Enter value for reprotno: 3**

**Enter value for damageamount: 67800**

```
old 1: insert into participated
values('&driverid','&regno','&reprotno','&damageamount')
new 1: insert into participated values('022','022','3','67800')
```

```
Enter value for driverid: 008
Enter value for regno: 008
Enter value for reprotno: 2
Enter value for damageamount: 2000
old 1: insert into participated
values('&driverid','&regno','&reprotno','&damageamount')
new 1: insert into participated values('008','008','2','2000')
```

#### SQL COMMANDS FOR QUERY:

- ➔ Query to update the damage amount for the car with a specific register number in the accident with report number between 1 & 200.

```
UPDATE PARTICIPATED SET DAMAGEAMOUNT=6235
WHERE REGNO=222 AND REPORTNO BETWEEN 1 AND 200;
```

- ➔ Query to find the total number of people who owned the cars that were involved in accidents in 2002.

```
SELECT COUNT (P.DRIVERID)
FROM PERSON P, PARTICIPATED PA, ACCIDENT A
WHERE P.DRIVERID=PA.DRIVERID AND PA.REPORTNO=A.REPORTNO
AND DDATE LIKE '%02';
```

- ➔ Query to find the number of accidents in which cars belonging to a specific model were involved.

```
SELECT COUNT (P.REPORTNO) FROM
ACCIDENT A, PARTICIPATED P, CAR C
WHERE A.REPORTNO=P.REPORTNO AND P.REGNO=C.REGNO
ORDER BY (MODEL);
```

**REPORT:**

- ➔ **Query to check if a person with a specific driver\_id has met with an accident in 2003.**

```
SELECT NAME
FROM PERSON P, PARTICIPATED PA, ACCIDENT A
WHERE P.DRIVERID = PA.DRIVERID AND
      PA.REPORTNO = A.REPORTNO AND
      A.DDATE LIKE '%02';
```

- ➔ **Query to display name of a person & the car he/she owns.**

```
SELECT P.NAME, C.MODEL
FROM PERSON P, CAR C, OWNS O
WHERE P.DRIVERID = O.DRIVERID AND
      O.REGNO = C.REGNO;
```

**RESULT:**

- ➔ **Output for the query to update the damage amount for the car with a specific register number in the accident with report number between 1 & 200.**

1 row updated

- ➔ **Output for the query to find the total number of people who owned the cars that were involved in accidents in 2002.**

**COUNT(P.DRIVERID)**

```
-----
      2
```

- ➔ **Output for the query to find the number of accidents in which cars belonging to a specific model were involved.**

**COUNT(P.REPORTNO)**

-----  
4

➡ **Output for the query to check if a person with a specific driver\_id has met with an accident in 2003.**

**NAME**

-----  
**ADARSH**

**1 row selected.**

➡ **Output for the query to display name of a person & the car he/she owns.**

<b>NAME</b>	<b>MODEL</b>
-----	-----
<b>ADARSH</b>	<b>SKODA</b>
<b>ANOOP</b>	<b>BMW</b>
<b>KIRAN</b>	<b>HYUNDAI</b>
<b>ABHISHEK</b>	<b>MARUTI</b>

**4 rows selected.**

## PROBLEM-2

Consider the following relations for an order processing database application in a company.

**CUSTOMER** (Cust #: int, Cname: string, City: string)

**ORDER** (Order #: int, Odate: date, Cust #: int, Ord-Amt: int)

**ORDER-ITEM** (Order #: int, Item #: int, qty: int)

**ITEM** (Item #: int, Unit Price: int)

**SHIPMENT** (Order #: int, Warehouse #: int, Ship-Date: date)

**WAREHOUSE** (Warehouse #: int, City: string)

- viii) Create the above tables by properly specifying the primary keys and the foreign keys.
- ix) Enter at least five tuples for each relation.
- x) Produce a listing: **CUSTNAME**, **NO\_OF\_ORDERS**, **AVG\_ORDER\_AMT**, where the middle column is the total number of orders by the customer and the last column is the average order amount for that customer.
- xi) List the **Order#** for the orders that were shipped from all the warehouses that the company has in a specific city.
- xii) Demonstrate how you delete **Item#** 10 from the **ITEM** table and make that field *null* in the **ORDER-ITEM** table.
- xiii) Generation of suitable reports.
- xiv) Create a suitable front end for querying and displaying the results.

## SCHEMA DESCRIPTION

The following schema describes an *Order Processing* database of a company. It describes the following relations: *Customer, Order, Order-Item, Item, Shipment and Warehouse*.

### Customer :

This relation contains necessary information about the customers of the company. It keeps track of each customer's I.D.(**CID**), his/her name (**Cname**) and the city (**City**) in which he/she lives.

ATTRIBUTE	TYPE
<b>CID</b>	<b>INTEGER</b>
<b>CNAME</b>	<b>VARCHAR (10) {String}</b>
<b>CITY</b>	<b>VARCHAR (10) {String}</b>

**Primary Key** : CID (Customer I.D)

**Constraints** : None

### Ordertab :

This relation maintains all the orders that have been placed by the customers of company. It keeps track of the Order No. (**ONO**), the date on which the order was placed (**Odate**), the I.D. of the customer who placed the order (**CID**) and the order amount (**Ord\_Amt**).

ATTRIBUTE	TYPE
<b>ONO</b>	<b>INTEGER</b>
<b>ODATE</b>	<b>DATE</b>
<b>CID</b>	<b>INTEGER</b>
<b>ORDER_AMT</b>	<b>INTEGER</b>

**Primary Key** : ONO (Order No.)

**Constraints** : The customer I.D.(**CID**) of this relation references the customer I.D. (**CID**) of the **Customer** relation. Hence **CID** acts as the foreign key for this relation.

### Itemtab:

This relation contains information about each item. It keeps track of the Item No. (**INO**) and the unit price (**Price**).

ATTRIBUTE	TYPE
INO	INTEGER
PRICE	INTEGER

**Primary Key** : INO (Item No.)

**Constraints** : None.

**Order item** :

This relation contains information about each order that has been placed. It keeps track of the Order No. (**ONO**), the Item No. (**INO**) and the quantity (**QTY**) of each item.

ATTRIBUTE	TYPE
ONO	INTEGER
INO	INTEGER
QTY	INTEGER

**Primary Keys** : ONO (Order No.), INO (Item No.)

**Constraints**: The Order No. (**ONO**) of this relation references the Order No. (**ONO**) of the relation **Ordertab**. Similarly the Item No. (**INO**) of this relation references the Item No. (**INO**) of the relation **Itemtab**. Hence **ONO** and **INO** act as the foreign keys for this relation.

**Warehouse** :

This relation contains information about the warehouses of the company. It keeps track of the Warehouse No. (**WareNo**) and the location (**City**) of each warehouse.

ATTRIBUTE	TYPE
WARENO	INTEGER
CITY	VARCHAR (10) {String}

**Primary Key** : WARENO (Warehouse No.)

**Constraints** : None.

**Shipment** :

This relation lists all the bulk consignments of each ordered item. It keeps track of the Order No. (**ONO**), the Warehouse No. (**WareNo**), the date (**Ship\_Date**) on which the consignment was shipped.

<b>ATTRIBUTE</b>	<b>TYPE</b>
<b>ONO</b>	<b>INTEGER</b>
<b>WARENO</b>	<b>INTEGER</b>
<b>SHIP_DATE</b>	<b>DATE</b>

**Primary Keys** : ONO (Order No.), WARENO (Warehouse No.)

**Constraints** : The Order No. (**ONO**) of this relation references the Order No. (**ONO**) of the relation **Ordertab**. Similarly the Warehouse No. (**WareNo**) of this relation references the Warehouse No. (**WareNo**) of the relation **WAREHOUSE**. Hence **ONO** and **WareNo** act as the foreign keys for this relation.

**D) TABLE CREATION**

```
CREATE TABLE CUSTOMER  
(CID INT,  
CNAME VARCHAR2(10),  
CITY VARCHAR2(10),  
PRIMARY KEY (CID));
```

```
CREATE TABLE ORDERTAB  
(ONO INT,  
ODATE DATE,  
CID INT,  
ORD_AMT INT,  
PRIMARY KEY (ONO),  
FOREIGN KEY (CID) REFERENCES CUSTOMER (CID));
```

```
CREATE TABLE ITEMTAB  
(INO INT,  
PRICE INT,  
PRIMARY KEY (INO));
```

```
CREATE TABLE ORDER_ITEM  
(ONO INT,  
INO INT,  
QTY INT,  
PRIMARY KEY (ONO,INO),  
FOREIGN KEY (ONO) REFERENCES ORDERTAB (ONO),  
FOREIGN KEY (INO) REFERENCES ITEMTAB (INO));
```

```
CREATE TABLE WAREHOUSE  
(WARENO INT,  
CITY VARCHAR2(10),  
PRIMARY KEY (WARENO));
```

```
CREATE TABLE SHIPMENT  
(ONO INT,  
WARENO INT,  
SHIPDATE DATE,  
PRIMARY KEY (ONO,WARENO),  
FOREIGN KEY (ONO) REFERENCES ORDERTAB (ONO),  
FOREIGN KEY (WARENO) REFERENCES WAREHOUSE (WARENO));
```

## **II) VALUE INSERTION**

### **CUSTOMER:**

```
INSERT INTO CUSTOMER VALUES('&CID','&CNAME','&CITY');
```

Eg.

```
INSERT INTO CUSTOMER VALUES('100', 'A', 'BAN');
```

### **ORDERTAB:**

```
INSERT INTO ORDERTAB VALUES  
('&ONO','&ODATE','&CID','&ORD_AMT');
```

Eg.

```
INSERT INTO ORDERTAB VALUES ('1000, '10-MAY-2002', '100', '10000');
```

### **ITEMTAB:**

```
INSERT INTO ITEMTAB VALUES('&INO','&PRICE');
```

Eg.

```
INSERT INTO ITEMTAB VALUES('10', '2000');
```

### **ORDER\_ITEM:**

```
INSERT INTO ORDER_ITEM VALUES('&ONO','&INO','&QTY');
```

Eg.

```
INSERT INTO ORDER_ITEM VALUES('1000', '10', '5');
```

### **WAREHOUSE:**

```
INSERT INTO WAREHOUSE VALUES('&WARENO','&CITY');
```

Eg.

```
INSERT INTO WAREHOUSE VALUES('15', 'SYD');
```

### **SHIPMENT:**

```
INSERT INTO SHIPMENT VALUES('&ONO','&WARENO','&SHIPDATE');
```

Eg.

```
INSERT INTO SHIPMENT VALUES('10000', '15', '10-JUN-2002');
```

## QUERIES

III)

Produce a listing: **CUSTNAME**, **NO\_OF\_ORDERS**, **AVG\_ORDER\_AMT**, where the middle column is the total number of orders by the customer and the last column is the average order amount for that customer.

### *Algorithm*

- The customer name can be obtained from the **CUSTOMER** relation.
- In order to obtain the number of orders per customer, it is required to join the **CUSTOMER** and **ORDERTAB** relations and then group by the **CID**.
- The average order amount can be obtained by considering the total cost per order. This can be achieved by taking the sum of the products of the **PRICE** and the **QUANTITY** of each item and then considering the average.

### *SQL Query:*

```
SELECT C.CID, C.CNAME, COUNT(*) AS NO_OF_ORDERS,
AVG(OI.QTY*I.PRICE)
FROM CUSTOMER C,ORDERTAB OT,ITEMTAB I,ORDER_ITEM OI
WHERE C.CID=OT.CID AND OT.ONO=OI.ONO AND OI.INO=I.INO
GROUP BY (C.CID,C.CNAME);
```

IV)

List the **Order#** for the orders that were shipped from all the warehouses that the company has in a specific city.

### *Algorithm:*

- The warehouse numbers of a specific city can be obtained from the **WAREHOUSE** relation. This can then be joined with the **SHIPMENT** relation on the warehouse attribute.

### *SQL Query:*

```
SELECT S.ONO
FROM SHIPMENT S, WAREHOUSE W
WHERE S.WARENO=W.WARENO AND W.CITY='SYD';
```

## VI) REPORT GENERATION

- i) We would like to know the order numbers of all orders placed by customers who belong to a particular city.

*Algorithm:*

- This can be obtained by first considering the CID's of all customers in the specified city from the **CUSTOMER** relation.
- This can then be combined with the **ORDERTAB** relation on the CID attribute.

*SQL QUERY:*

```
SELECT O.ONO
FROM CUSTOMER C, ORDERTAB O
WHERE C.CID=O.CID AND C.CITY='SYD';
```

- ii) We would like to know the details of all orders that the company has received.

*Algorithm :*

- The details of each order is clearly maintained in the **ORDERTAB** relation

*SQL Query:*

```
SELECT *
FROM ORDERTAB;
```

**RESULT**

SELECT \* FROM CUSTOMER;

CID	CNAME	CITY
----	-----	-----
100	A	BAN
200	B	MAD
300	C	DEL
400	D	LUC
500	E	BOM

SELECT \* FROM ORDERTAB;

ONO	ODATE	CID	ORD_AMT
-----	-----	----	-----
1000	10-MAY-02	100	10000
2000	11-MAY-03	200	30000
3000	12-MAY-02	100	12500
4000	13-MAY-03	200	35000
5000	14-MAY-02	500	7500

SELECT \* FROM ITEMTAB;

INO	PRICE
----	-----
10	2000
20	3000
30	2500
40	3500
50	1500

SELECT \* FROM ORDER\_ITEM;

ONO	INO	QTY
-----	-----	-----

-----	-----	-----
1000	10	5
2000	20	10
3000	30	5
4000	40	10
5000	50	5

SELECT \* FROM WAREHOUSE;

WARENO	CITY
-----	-----
15	SYD
25	SYD
35	BOM
45	BOM
55	DEL

SELECT \* FROM SHIPMENT;

ONO	WARENO	SHIPDATE
-----	-----	-----
1000	15	16-JUN-02
2000	25	17-JUL-03
3000	35	18-AUG-02
4000	45	19-SEP-03
5000	55	20-OCT-02

### ***QUERY RESULTS:***

iii)

```
SELECT C.CID, C.CNAME, COUNT(*) AS NO_OF_ORDERS,
AVG(OI.QTY*I.PRICE)
FROM CUSTOMER C,ORDERTAB OT,ITEMTAB I,ORDER_ITEM OI
WHERE C.CID=OT.CID AND OT.ONO=OI.ONO AND OI.INO=I.INO
GROUP BY (C.CID,C.CNAME);
```

CID	CNAME	NO_OF_ORDERS	AVG(OI.QTY*I.PRICE)
-----	-----	-----	-----

100	A	2	11250
200	B	2	32500
500	E	1	7500

iv)

```
SELECT S.ONO
FROM SHIPMENT S, WAREHOUSE W
WHERE S.WARENO=W.WARENO AND W.CITY='SYD';
```

```
ONO
-----
1000
2000
```

**REPORT RESULTS:**

- i) We would like to know the order numbers of all orders placed by customers who belong to a particular city.

```
SELECT O.ONO
FROM CUSTOMER C, ORDERTAB O
WHERE C.CID=O.CID AND C.CITY='BAN';
```

```
ONO
-----
1000
3000
```

- ii) We would like to know the details of all orders that the company has received.

```
SELECT *
FROM ORDERTAB;
```

ONO	ODATE	CID	ORD_AMT
-----	-----	----	-----
1000	10-MAY-02	100	10000
2000	11-MAY-03	200	30000

3000	12-MAY-02	100	12500
4000	13-MAY-03	200	35000
5000	14-MAY-02	500	7500

## **PROBLEM 3**

Consider the following database of student enrollment in course and books adopted for each course .

**STUDENT** (regno:string, name:string, major:string, bdate:date)

**COURSE** (course#:int, cname:string, dept:string)

**ENROLL** (regno:string, course#:int, sem:int, marks:int )

**BOOK\_ADOPTION** (course#:int., sem:int, book-ISBN:int)

**TEXT**(book-ISBN:int, book-title:string, publisher:string, authour:string).

- viii) Create the above tables by properly specifying the primary keys and the foreign keys .
- ix) Enter atleast five tuples for each relation.
- x) Demonstrate how you add a new text book to the database and make this book be adopted by some department.
- xi) Produce a list of text books (include Course #,Book-ISBN, Book-title) in the alphabetical order for courses offered by the 'CS' department that use more than two books.
- xii) List any department that has all its adopted books published by a specific publisher.
- xiii) Generation of suitable reports.
- xiv) Create suitable front end for quering and displaying the results.

**TABLE STUDENT**

**REGNO** is of datatype CHAR of length 10 . It is a PRIMARY KEY which is NOT NULL. It is the REGISTRATION NUMBER of each student.

**NAME** is of datatype VARCHAR of length 10. It is a SECONDARY KEY which can be NULL . It is the NAME of each student .

**MAJOR** is of datatype VARCHAR of length 10. It is a SECONDARY KEY which can be NULL. It specifies the MAJOR SUBJECT that you want to be specialised in.

**BDATE** is of datatype date .It is a SECONDARY KEY which can be NULL. It gives the BIRTH DATE of the specified student .

**TABLE COURSE**

**COURSE** is of datatype int .It is the PRIMARY KEY which is NOT NULL.It is the COURSE NUMBER of each subject.

**CNAME** is of datatype varchar of length 20.It is a SECONDARY KEY which can be NULL.It specifies the COURSE NAME of that subject.

**DEPT** is of datatype VARCHAR of length 10.It is a SECONDARY KEY which can be NULL.It is the DEPARTMENT NAME which offers the above course.

**TABLE ENROLL**

**REG\_NO** is of datatype CHAR of length 10.It is a FOREIGN KEY that references the table STUDENT which on delete cascades.

**COURSE** is of datatype INT. It is a FOREIGN KEY that references the table COURSE which on delete cascades.

**SEM** is of datatype INT. It is a SECONDARY KEY which can be NULL.It gives the SEMESTER of that student.

REGNO and COURSE\_NO together acts as a PRIMARY KEY.

**TABLE TEXT**

**BOOKISBN** is of datatype INT. It is a PRIMARY KEY which is NOT NULL.It gives the ISBN NUMBER of each book that a student has to buy.

**BOOKTITLE** is of datatype VARCHAR of length 10.It is a SECONDARY KEY which can be NULL.It specifies the TITLE of each book.

**PUBLISHER** is of datatype VARCHAR of length 10.It is a SECONDARY KEY which can be NULL. It gives the PUBLISHER NAME of the specified book.

**AUTHOR** is of datatype VARCHAR of length 10.It is a SECONDARY KEY which can be NULL.It specifies the AUTHOR NAME of the given book.

**TABLE BOOK\_ADOPTION**

**COURSE** is of datatype INT. It is a FOREIGN KEY that references the table COURSE which on delete cascades.

**SEM** is of datatype INT. It is a SECONDARY KEY which can be NULL.It gives the SEMESTER of the student who has borrowed the book.

**BOOK\_ISBN** is of datatype INT.It is a FOREIGN KEY that references the table TEXT which on delete cascades.

**COURSE\_NO** acts also as a PRIMARY KEY.

**TABLE CREATION:**

```
CREATE TABLE STUDENT
(
  REGNO VARCHAR(30) PRIMARY KEY,
  NAME VARCHAR(30) NOT NULL,
  MAJOR VARCHAR(30) NOT NULL,
  BDATE DATE NOT NULL
);
```

```
CREATE TABLE COURSE
(
  COURSE INTEGER PRIMARY KEY,
  CNAME VARCHAR(30) NOT NULL,
  DEPT VARCHAR(30) NOT NULL
);
```

```
CREATE TABLE ENROLL
(
  REG_NO VARCHAR(30) NOT NULL,
  COURSE INTEGER NOT NULL,
  SEM INTEGER NOT NULL,
  MARKS INTEGER NOT NULL,
  PRIMARY KEY(REG_NO,COURSE,SEM),
  FOREIGN KEY (REG_NO) REFERENCES STUDENT(REGNO) DELETE ON
  CASCADE,
  FOREIGN KEY (COURSE) REFERENCES COURSE(COURSE) DELETE ON
  CASCADE
);
```

```
CREATE TABLE TEXT
(
  BOOKISBN INTEGER PRIMARY KEY,
  BOOKTITLE VARCHAR(30) NOT NULL,
  PUBLISHER VARCHAR(30) NOT NULL,
  AUTHOR VARCHAR(30) NOT NULL
);
```

```
CREATE TABLE BOOK_ADOPTION
(
```

```

COURSE INTEGER NOT NULL,
SEM INTEGER NOT NULL,
BOOK_ISBN INTEGER NOT NULL,
PRIMARY KEY(COURSE_,SEM,BOOK_ISBN),
FOREIGN KEY (COURSE) REFERENCES COURSE(COURSE) DELETE ON
CASCADE,
FOREIGN KEY(BOOK_ISBN) REFERENCES TEXT(BOOKISBN) DELETE ON
CASCADE
);

```

```

INSERT INTO STUDENT VALUES('1BM02IS012','ANSHUMAN
ATRI','DATABASE','15-JAN-84');
INSERT INTO STUDENT VALUES('1BM02CS012','A MNC','DMS','25-FEB-84');
INSERT INTO STUDENT VALUES('1BM02TC012','TELE TECHNO','SSDT','11-
DEC-84');
INSERT INTO STUDENT VALUES('1BM02EE012','ELCTRA','POWER
GENERATION','1-APR-84');
INSERT INTO STUDENT VALUES('1BM02EC012','GG','POWER
ELECTRONICS','5-NOV-84');

```

```

INSERT INTO COURSE VALUES(1,'DATABASE','IS');
INSERT INTO COURSE VALUES(2,'DMS','IS');
INSERT INTO COURSE VALUES(3,'SSDT','TC');
INSERT INTO COURSE VALUES(4,'POWER GENERATION','EE');
INSERT INTO COURSE VALUES(5,'POWER ELECTRONICS','EC');

```

```

INSERT INTO TEXT VALUES(1,'DATABASE A SYSTEMATIC
APPROACH','JOHN WILEY','R ASHOK KUMAR');
INSERT INTO TEXT VALUES(2,'DMS FOR
DUMMIES','PEARSON','MADHUPRIYA');
INSERT INTO TEXT VALUES(3,'SSDT NO ONE CAN TEACH
BETTER','PEARSON','GAURA');
INSERT INTO TEXT VALUES(4,'POWER GENERATION
BIBLE','TMH','MEENA');
INSERT INTO TEXT VALUES(5,'POWER OF POWER ELECTRONICS','O
REILLY','GG THE GREAT');

```

```

INSERT INTO ENROLL VALUES('1BM02IS012',1,5,98);
INSERT INTO ENROLL VALUES('1BM02CS012',2,3,88);
INSERT INTO ENROLL VALUES('1BM02TC012',3,5,88);
INSERT INTO ENROLL VALUES('1BM02EE012',4,5,88);

```

```
INSERT INTO ENROLL VALUES('1BM02EC012',5,5,88);
```

```
INSERT INTO BOOK_ADOPTION VALUES(1,5,1);
INSERT INTO BOOK_ADOPTION VALUES(2,3,2);
INSERT INTO BOOK_ADOPTION VALUES(3,5,3);
INSERT INTO BOOK_ADOPTION VALUES(4,5,4);
INSERT INTO BOOK_ADOPTION VALUES(5,5,5);
```

```
SELECT C.COURSE,T.BOOKISBN,T.BOOKTITLE FROM COURSE
C,BOOK_ADOPTION BA,TEXT T WHERE C.COURSE=BA.COURSE_ AND
BA.BOOK_ISBN=T.BOOKISBN AND C.DEPT='CS' AND (SELECT
COUNT(COURSE_) FROM BOOK_ADOPTION WHERE
COURSE_=C.COURSE GROUP BY COURSE_
HAVING COUNT(COURSE_) >2)>2 ORDER BY T.BOOKTITLE;
```

```
INSERT INTO TEXT VALUES(7,'DATABASE FOR DUMMIES','PEARSON','R
A K');
INSERT INTO TEXT VALUES(6,'FUNDAMENTALS OF DATABASE','JOHN
WILEY','R ASHOK KUMAR');
INSERT INTO BOOK_ADOPTION VALUES(1,5,6);
INSERT INTO BOOK_ADOPTION(1,5,7);
```

```
SELECT C.DEPT,T.PUBLISHER FROM COURSE C,TEXT
T,BOOK_ADOPTION BA WHERE C.COURSE=BA.COURSE_ AND
T.BOOKISBN=BA.BOOK_ISBN AND
T.PUBLISHER = ALL ( SELECT T1.PUBLISHER FROM COURSE
C1,BOOK_ADOPTION BA1,TEXT T1 WHERE
BA1.BOOK_ISBN=T1.BOOKISBN AND
BA1.COURSE_=C1.COURSE AND C.DEPT=C1.DEPT);
```

```
INSERT INTO COURSE VALUES(6,'ADVANCE MP','CS');
INSERT INTO STUDENT VALUES('1BM02CS007','HARDWARE
PRO','MP','21-OCT-84');
INSERT INTO TEXT VALUES (9,'MP FOR DUMMIES','PEARSON','A K
ROY');
INSERT INTO TEXT VALUES (10,'A TO Z OF
MP','PEARSON','BURCHANDI');
INSERT INTO TEXT VALUES (8,'MP INTEL FAMILY','PEARSON','BARRY B
BREY');
INSERT INTO BOOK_ADOPTION VALUES(6,5,8);
INSERT INTO BOOK_ADOPTION VALUES(6,5,9);
INSERT INTO BOOK_ADOPTION VALUES(6,5,10);
```

**QUERIES****(i)**

Demonstrate how you add a new text book to the database and make this book be adopted by some department

**Query**

```
INSERT INTO TEXT VALUES ('1111','MULTIMEDIA','PHI','NAVATHE');
INSERT INTO BOOK_ADOPTION VALUES('61','6','1111');
```

**Explanation**

INSERT can be used to add a single tuple to a relation. Thus a information about a new text book can be added to the TEXT entity using INSERT command. The new text book can be made to be adopted by some department using INSERT. The values which are added to the BOOK\_ADOPTION contains COURSENO and SEM of the department and semester which has uses the textbook, along with the BOOK\_ISBN of the textbook through which other information of the textbook can be obtained.

**(ii)** Produce a list of textbooks in the alphabetic order for courses offered by the 'CS' department that use more than two books

**Query**

```
SELECT C.COURSENO, T.BOOK_ISBN, T.BOOK_TITLE
FROM COURSE C, BOOK_ADOPTION BA, TEXT T
WHERE C.COURSENO = BA.COURSENO AND C.DEPT = 'CS'
      AND EXISTS ( SELECT COUNT(COURSENO)
                   FROM BOOK_ADOPTION
                   WHERE COURSENO = C.COURSENO
```

```

GROUP BY COURSENO
HAVING COUNT(COURSENO)>2)
ORDER BY BOOK_TITLE

```

### Explanation

The first nested query selects the Departments which use more than two books. To select the departments EXIST function is used to check whether there are any department which use more than two books. In the outer query, we select the courseno, book\_isbn and book\_title of the textbooks for courses offered by the 'CS' department. In the outer query, the join condition C.COURSENO= BA.COURSENO relates the COURSE and BOOK\_ADOPTION and the condition DEPT= 'CS' is a selection condition. ORDER-BY clause is used to order the tuples in the result of query according to the BOOK\_TITLE

- (iii) List any department that has all its adopted books published by a specific Publisher.

### Query

```

SELECT DISTINCT C.DEPT, T.PUBLISHER
FROM COURSE C,TEXT T, BOOK_ADOPTION BA
WHERE C.COURSENO = BA.COURSENO AND
T.BOOK_ISBN = BA.BOOK_ISBN AND
T.PUBLISHER = 'TATAMC' AND
T.PUBLISHER = ALL(SELECT T1.PUBLISHER
FROM COURSE C1,
BOOK_ADOPTION BA1,TEST T1
WHERE
BA1.BOOK_ISBN=T1.BOOK_ISBN
BA1.COURSENO=C1.COURSENO
AND C.DEPT=C1.DEPT);

```

### Explanation

The nested query selects publishers from relations COURSE, BOOK\_ADOPTION and TEST using the join conditions BA1.BOOK\_ISBN=T1.BOOK\_ISBN which relates book adoption to the text entity, BA1.COURSENO=C1.COURSENO which relates BOOK\_ADOPTION and COURSE entities. The keyword ALL returns true if T.PUBLISHER value is equal to all the results

of the nested query. The department name and the publisher whose books have been adopted by the department are projected from the entities mentioned in the FROM clause which are joined using the join conditions given in the WHERE clause.

## **RESULT**

The result of the second query is

COURSENO	BOOK_ISBN	BOOK_TITLE
61	1111	MULTIMEDIA
63	4444	NETWORKS

The result of third query is

DEPT	PUBLISHER
IS	PEARSON EDUCATION
CS	PHI
IT	TATA MCGRAW HILL

## GENERATION OF REPORTS

WE WOULD LIKE TO SEE THE NUMBER OF STUDENTS FOR EACH COURSE

```
SELECT COUNT(REGNO),C.COURSENO
FROM COURSE C,STUDENT S,ENROLL E
WHERE E.REGNO=S.REGNO AND C.COURSENO=E.COURSENO
GROUP BY COURSENO
```

### **Explanation**

In the above query entities COURSE,STUDENT, ENROLL are joined using the join condition E.REGNO=S.REGNO which relates ENROLL and STUDENT and the join condition C.COURSENO=E.COURSENO which relates ENROLL and COURSE. The number of students are calculated using COUNT operation.

WE WOULD LIKE TO SEE NAME AND USN OF THE STUDENTS OF A SPECIFIC COURSE

```
SELECT S.REGNO,S.NAME
FROM STUDENT S,ENROLL E,COURSE C
WHERE S.REGNO=E.REGNO AND
E.COURSENO=C.COURSENO
```

In the above query entities COURSE,STUDENT, ENROLL are joined using the join condition E.REGNO=S.REGNO which relates ENROLL and STUDENT and the join condition C.COURSENO=E.COURSENO which relates ENROLL and COURSE. The REGNO and NAME is selected to see name and usn of the students of the specific course .

## **PROGRAM -4**

Consider the following relations for the details maintained by a book dealer.

**AUTHOR** (Author-id: int, Name: string, City: string, Country: string)

**PUBLISHER** (Publisher-id: int, Name: string, City: string, Country: string)

**CATALOG** (Book-id: int, title: string, author-id: int, Publisher-id: int, Category-id: int, Year: int, Price: int)

**CATEGORY** (Category-id: int, Description: string)

**ORDER-DETAILS** (Order-no : int, Book-id: int, Quantity: int)

1. Create the above tables by properly specifying the primary keys and the foreign keys.
2. Enter at least five tuples for each relation.
3. Give the details of the authors who have 2 or more books in the catalog and the price of the books is greater than the average price of the books in the catalog and the year of publication is after 2000.
4. Find the author of the book which has maximum sales.
5. Demonstrate how you increase the price of books published by a specific publisher by 10%.
6. Generation of suitable reports.
7. Create a suitable front end for querying and displaying the results.

## SCHEMA DESCRIPTION

The following schema describes a database maintained by a Book-Dealer. It describes the following relations: *Author*, *Publisher*, *Catalog*, *Category*, *Order-Details*.

### Author:

This relation contains necessary information about the authors. It keeps track of each author's I.D.(**AID**), his/her name (**ANAME**), the city (**ACITY**) in which he/she lives, and the country (**ACOUNTRY**) to which he/she belongs.

ATTRIBUTE	TYPE
<b>AUTHOR_ID</b>	<b>INTEGER</b>
<b>ANAME</b>	<b>VARCHAR (10) {String}</b>
<b>ACITY</b>	<b>VARCHAR (10) {String}</b>
<b>ACOUNTRY</b>	<b>VARCHAR(10) {String}</b>

**Primary Key** : AID (Author\_I.D)

**Constraints** : None

### Publisher:

This relation maintains all the details of the publisher. It contains the publisher-id (**PID**), name of the publisher (**PNAME**), the city in which the publisher lives (**PCITY**) and the country in which the publisher lives (**PCOUNTRY**).

ATTRIBUTE	TYPE
<b>PID</b>	<b>INTEGER</b>
<b>PNAME</b>	<b>VARCHAR (10) {String}</b>
<b>PCITY</b>	<b>VARCHAR (10) {String}</b>
<b>PCOUNTRY</b>	<b>VARCHAR (10) {String}</b>

**Primary Key** : PID (Publisher's I.D)

**Constraints** : None

**Category:**

This relation category contains the category-i.d (**CID**) along with the description for all the books (**DESCR**).

ATTRIBUTE	TYPE
<b>CID</b>	<b>INTEGER</b>
<b>DESCR</b>	<b>VARCHAR (10) {String}</b>

**Primary Key** : CID (Category I.D)

**Constraints** : None.

**Catalog :**

This relation contains information about all the books i.e. book-id (**BID**), title (**BTITLE**), along with the details of the author of the respective books i.e. author-Id (**AID**), publisher (**PID**) and other details of the books such as category-id (**CID**), ,price (**PRICE**), and year (**YEAR**).

ATTRIBUTE	TYPE
<b>BID</b>	<b>INTEGER</b>
<b>BTITLE</b>	<b>VARCHAR (10) {String}</b>
<b>AID</b>	<b>INTEGER</b>
<b>PID</b>	<b>INTEGER</b>
<b>CID</b>	<b>INTEGER</b>
<b>PRICE</b>	<b>INTEGER</b>
<b>YEAR</b>	<b>INTEGER</b>

**Primary Keys** : BID (Book I.D.)

**Constraints:** The Author I.d (**AID**) of this relation references the Author-i.d (**AID**) of the **Author** relation. Similarly the Publisher id (**PID**) of this relation references the Publisher id (**PID**) of the relation **Publisher**. Also the Category- i.d (**CID**) of this relation references the Category-i.d (**CID**) of the relation **Category**.

**Order-Details :**

This relation contains information about all the orders that have been placed i.e. Order-No (**ONO**), book-id (**BID**) and the quantity (**QTY**).

ATTRIBUTE	TYPE
<b>ONO</b>	<b>INTEGER</b>
<b>BID</b>	<b>INTEGER</b>
<b>QTY</b>	<b>INTEGER</b>

**Primary Key** : ONO (Order No.), BID (Book id.)

**Constraints** : The Book-id (**BID**) of this relation references the Book-id (**BID**) of the Catalog relation.

**TABLE CREATION:**

1.

```
create table author(
  2 author_id int primary key,
  3 author_name varchar(15)
  4 ,city varchar(15),
  5 country varchar(15));
```

Table created.

2.

```
create table publisher(
  2 publisher_id int primary key,
  3 publisher_name varchar(15),
  4 city varchar(15),
  5 country varchar(15));
```

Table created.

3.

```
create table category(
  2 category_id int primary key,
  3 description varchar(15));
```

Table created.

4.

```
create table catalog(
  2 book_id int primary key,
  3 tittle varchar(15),author_id int,p_id int,c_id int,
  4 year int,price int,
  5 foreign key(author_id) references author(author_id),
  6 foreign key(p_id) references publisher(publisher_id),
  7 foreign key(c_id) references category(category_id));
```

Table created.

5.

```
create table order_details(
  2 order_no int primary key,
  3 book_id int,
```

4 quantity int,  
 5 foreign key(book\_id) references catalog(book\_id));  
 Table created.

### **INSERTION OF VALUES INTO THE TABLES:**

**insert into author values (222,'T ASHA','MANDYA','INDIA');**  
**insert into PUBLISHER VALUES (100,'McGrawHill','New Delhi','Hindustan')**  
**insert into CATEGORY values (101,'SYS S/W')**  
**insert into catlog values (101,'LOGIC DESIGN',111,100,101,2000,5500);**  
**insert into order\_details values('&order\_no','&book\_id ','&quantity');**

## **ALGORITHM FOR QUERIES**

### **QUERY III**

- This query gives details of the authors who have 2 books in the catalog and the price of the books is greater than the average price of the books in the catalog and the year of publication is after 2000.

### **SQL COMMAND**

```
select a.auth_id as AI, name as NAM, city, count (*) as CNT
from author a, catalog c
where a.auth_id=c.auth_id and year>2000and a.auth_id
in (select auth_id from catalog where price >
      (select avg (price) from catalog) )
group by (a.auth_id, name, city) having count(*)>=2;
```

## ▪ ALGORITHM

Step1: select average price of all books from catalog relation.

Step2: select author id from catalog whose book(s) price is greater than the price retrieved from step1.

Step3: select author id,author name,city and count of books from author, catalog and rename them as AI,NAM,city,CNT respectively where author id in author relation is equal to author id in catalog relation and the year of publication is after 2000 and that author id exists in the set retrieved from step2 and group the result based on author id ,author name and city having the count of books greater than or equal to 2.

## QUERY IV

**This query demonstrates how to retrieve the author of the book which has maximum sales.**

### ▪ SQL COMMAND

```

select distinct a.name
from author a, catalog c,order_details odm
where a.auth_id=c.auth_id and odm.b_id=c.book_id
and exists ( select od.b_id,sum(od.quantity)
from order_details od
where od.b_id=odm.b_id
group by b_id

```

```

having sum(od.quantity)>=all(
select sum(quantity)
from order_details
group by b_id ) );

```

### ▪ ALGORITHM

*Step 1: Select sum of quantity of sales of each book based on book id (i.e. group based on book id) from order\_details relation.*

Step 2: In turn select the book id and the quantity of sale of the book from the order\_details relation, which exists in the result of step1 and is the greatest among them.

Step 3: Retrieve author name from author, catalog, order\_details relations where author id in author relation is equal to that of catalog relation and book id in catalog is equal to book id in order\_details and the result in step2 is satisfied.

## QUERY V

This query demonstrates how to increase the price of the books published by a specific publisher by 10%.

### ▪ SQL COMMAND

```

select pub_id, pname, 1.1*price as new_price
from publisher, catalog
where pub_id=p_id and pname='aaa';

```

▪ ALGORITHM

Step 1: select the books whose publisher name is 'aaa' from publisher.

Step 2: select the tuples from catalog whose publisher id is same as that of in the step1 result.

Step 3: Increase the price of all the books in the result of step 2 by 10%.

Step 4: Retrieve the publisher's id, name and new price of the books from the result of step3.

(Note: In the above nested queries the inner one gets evaluated first).

## **REPORT**

**In this project we are mainly dealing with maintenance of books by a book dealer. This schema mainly includes relations like author, publisher,**

**catalog, order\_details etc.**

The queries which we have implemented includes,

- Demonstration on how to retrieve the details of the authors who have 2 or more books in the catalog and their price is greater than the average price of all books in the catalog and the year of publication is after 2000.
- Demonstration on how to find the author with the maximum sales.
- Demonstration on how to increase the price of books published by a specific publisher.

The SQL reports generation code for these queries are as follows,

### **QUERY III**

```
select a.auth_id as ai,name as nam,city,count(*) as cnt
from author a,catalog c where a.auth_id=c.auth_id and year>2000 and a.auth_id
in (select auth_id from catalog where price >
(select avg(price) from catalog))
group by(a.auth_id,name,city)
having count(*)>=2;
```

**QUERY IV**

```

select distinct a.name from author a,catalog c,order_details odm
where a.auth_id=c.auth_id and odm.b_id=c.book_id and
exists (
select od.b_id,sum(od.quantity) from order_details od where
od.b_id=odm.b_id group by b_id having sum(od.quantity)>=all
(select sum(quantity) from order_details group by b_id)
);

```

**QUERY V**

```

select pub_id, pname, 1.1*price as new_price
from publisher, catalog
where pub_id=p_id and pname='aaa';

```

```

Ttitle LEFT 'AUTHOR REPORT' skip 2
Btitle center 'GooD Day' right 'Page >>====>'SQL.PNO
skip 3
SET UNDERLINE ~

```

```

Column author_id HEADING 'AUTHOR | ID No.'
Column name HEADING 'AUTHOR | Name'
Column title HEADING 'BooK | Title'
Column description HEADING 'Description'
Column book_id HEADING 'BooK | ID No.'
Column year HEADING 'Published |
Year'
Column price HEADING 'Cost' format
$99,999.99

```

break on author\_id skip page

compute sum of price on author\_id

```
select a.author_id,a.name,
       c.title,cat.description,c.book_id,c.year,c.price
from author a,catlog c,category cat
where a.author_id=c.a_id and cat.category_id=c.c_id
ORDER BY c.a_id;
```

clear computes

clear columns

clear breaks

ttitle off

btitle off

```
Title  CENTER 'PUBLISHER REPORT' skip 2
Btitle center 'HAVE A NICE DAY :-)' right 'Page
>>====>'SQL.PNO skip 3
SET UNDERLINE *
```

```
Column PUBLISHER_id HEADING 'PUBLISHER|ID
No.'
```

```
Column name          HEADING 'AUTHOR|Name'
```

```
Column title         HEADING 'Book|Title'
```

```
Column description  HEADING 'Description'
```

```
Column NAME         HEADING 'PUBLISHER|Name'
```

```
Column year         HEADING 'Published|Year'
```

```
Column price        HEADING 'Cost' format
```

```
$99,999.99
```

break on PUBLISHER\_ID skip page

select

```
P.PUBLISHER_id,a.name,c.title,cat.description,p.name,c.y
ear,c.price
```

```
from author a,catlog c,category cat,PUBLISHER P
```

```
where P.PUBLISHER_id=c.P_id and
```

```
cat.category_id=c.c_id AND a.author_id=c.a_id
```

```
ORDER BY P.PUBLISHER_id;
```

clear computes

clear columns

clear breaks

ttitle off

btitle off

**RESULT**

**These are the contents of each relation after insertion command.**

select \* from author ;

AUTH_ID	NAME	CITY	COUNTRY
1001	navathe	bangalore	india
1002	beck	rome	italy
1003	o-broien	paris	france
1999	godse	bangalore	india
9999	grewal	delhi	India

5 rows selected.

*select \* from publisher ;*

PUB_ID	PNAME	CITY	COUNTRY
123	Mcgraw	london	england
345	shubhas	mangalore	india
567	shiva	bangalore	india
789	manu	manipur	india
990	aaa	delhi	India

5 rows selected.

select \* from order\_details;

ORDER_NO	B_ID	QUANTITY
111	10	3
222	20	4
333	30	9
999	30	7
888	50	8
666	60	1

6 rows selected.

```
select * from catalog;
```

BOOK_ID	TITLE	AUTH_ID	P_ID	CAT_ID	YEAR
10 890	rdbms	1001	123	100	1990
20 240	c++	1001	345	200	2003
30 600	let us c	1003	567	300	2004
40 860	java2	1999	990	400	2002
50 780	linux	1999	789	500	2001
60 200	oops with c++	1002	990	400	2000

6 rows selected.

```
select * from category;
```

CAT_ID	DSCR
100	aaa
200	bbb
300	ccc
400	eee
500	ddd

5 rows selected.

This is the result of query iii executed using the procedure as shown in the

report. It gives the details of the author who has 2 books in the catalog and their price is greater than all the books in the catalog and the year of publication is after 2000.

### QUERY III

AI	NAM	CITY	CNT
1999	nanda	bangalore	2

This is the result of query iv executed using the procedure as shown in the report. It gives the author name who has maximum sales.

### QUERY IV

NAME

o-broien

This is the result of query v executed using the procedure as shown in the report. it gives the publisher id ,name and the new price of the books which is increased by 10% published by 'aaa';

### QUERY V

PUB_ID	PNAME	NEW_PRICE
990	aaa	946
990	aaa	220

@d:\oracle\ora92\bin\rep1.sql;

### PUBLISHER REPORT

PUBLISHER ID No.	PUBLISHER Name	Book Title	Book Description	PUBLISHER Name	PUBLISHER Year	Published Cost
100	B N SHOBHA	LOGIC DESIGN	SYS S/W	McGrawHill	2000	\$5,500.00
	T ASHA	FAFL	SYS S/W	McGrawHill	2000	\$2,575.00
	V VANI	DC	Network	McGrawHill	2004	\$3,755.45

## PUBLISHER REPORT

PUBLISHER ID No.	PUBLISHER Name	Book Title	Book Description	PUBLISHER Name	PUBLISHER Year	Published Cost
200	B N SHOBHA	COMP ORG	Electronics	Pearson	2001	\$3,750.00
	R NAGARAJ	DBMS	Applications	Pearson	2002	\$999.99
	V VANI	CN	Network	Pearson	2005	\$1,000.10
	B N SHOBHA	USP	SYS S/W	Pearson	2004	\$5,050.55

HAVE A NICE DAY :-)

Page &gt;&gt;====&gt; 2

## PUBLISHER REPORT

PUBLISHER ID No.	PUBLISHER Name	Book Title	Book Description	PUBLISHER Name	PUBLISHER Year	Published Cost
300	T ASHA	OS	SYS S/W	Prentice Hall	2000	\$4,250.75
	R NAGARAJ	ADA	Network	Prentice Hall	2003	\$1,550.95
	T ASHA	SS	SYS S/W	Prentice Hall	2005	\$500.50

PUBLISHER ID No.	PUBLISHER Name	Book Title	Book Description	PUBLISHER Name	PUBLISHER Year	Published Cost
400	R NAGARAJ	Data Structres	Applications	Addison	1998	\$1,050.25
	B N SHOBHA	MC	Applications	Addison	2005	\$5,750.65

## PROBLEM – 5

Consider the following database for a banking enterprise

**BRANCH** (branch\_name: string, branch\_city: string, assets: real)

**ACCOUNT** (accno: int, branch\_name: string, balance: real)

**CUSTOMER** (customer\_name: string, customer\_street: string, city: string)

**DEPOSITOR** (customer\_name: string, accno: int)

**LOAN** (loan\_number: int, branch\_name: string, amount: real)

**BORROWER** (customer\_name: string, loan\_number: int)

- viii) Create the above tables by properly specifying the primary keys and the foreign keys.
- ix) Enter atleast five tuples for each relation.
- x) Find all the customers who atleast two accounts at the **MAIN** branch.
- xi) Find all the customers who have an account at **all** branches located in a specific city.
- xii) Demonstrate how you delete all account tuples at every branch located in a specific city.
- xiii) Generation of suitable reports.
- xiv) Create suitable front end for querying and displaying the results.

## SCHEMA DESCRIPTION

This database is created for a banking enterprise. This database keeps an account of all the branches in a particular city, information about the customer like details of the customer, in which branch his account exists and details of loan taken from the customer(if any).

Table name: **BRANCH**

This table consists of details of the branch in a particular city. It consists of the following attributes:

**branch name**: It gives the name of the branch. Since no two branches of a city can have the same name it is taken as the **primary key** of this table.

**branch city**: It gives the name of the city where the particular branch is located.

**assets**: It gives the assets of the particular branch.

Table name: **ACCOUNT**

This table gives the details of the each account of a particular like the account number, balance in the account.

**accno**: It specifies the account number. Since no two account numbers can be same it is taken as the **primary key** of this table.

**branch name**: It gives the name of the branch in which the account is existing.

This attribute refers the **BRANCH** table to get the name of the branch, hence it is considered to be the **foreign key**.

**balance**: This gives the balance present in the corresponding account number.

Table name: **CUSTOMER**

This table customer details. The following attributes are present in this table:

**customer name**: This gives the name of the customer. Two customers cannot have same name, therefore it is considered to be the **primary key**.

**customer street**: This gives the name of the street given by the customer.

**city**: This gives the name of the city in which the customer lives.

Table name: **DEPOSITOR**

This table gives details of the account number of a particular customer.

**customer name**: This gives the name of the customer. It refers **CUSTOMER** table to get the name, therefore it is considered as the **foreign key**.

**accno**: It gives the account number of the customer and refers the **ACCOUNT** table to get the accno, therefore it is considered as the **foreign key**.

Since a customer can have more than one account and accno can be same in different branches we consider both the attributes as the **primary key**.

Table name: **LOAN**

This table gives the details of the loan sanctioned in the branches.

**loan number**: This gives the loan number. No two loan numbers can be same, therefore it is considered as the **primary key**.

**branch name**: It gives the name of the branch in which the account is existing.

This attribute refers the **BRANCH** table to get the name of the branch, hence it is considered to be the **foreign key**.

**amount**: This attribute gives the amount for which the loan is sanctioned.

Table name: **BORROWER**

This table gives the details of which customer as taken the loan.

**customer\_name**: This gives the name of the customer. It refers **CUSTOMER** table to get the name, therefore it is considered as the **foreign key**.

**loan\_number**: This gives the loan number. It refers **LOAN** table to obtain the loan number, therefore it is taken as the **foreign key**.

Here both the attributes are together taken as the **primary key**.

## **CREATION OF TABLES AND INSERTION OF TUPLES**

The CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first, each attribute is given a name, a data type to specify its domain of values and any attribute constraints. The key, entity integrity, referential integrity constraints can be specified within the CREATE TABLE.

INSERT is used to add a single tuple into the relation. We must specify the relation name and a list of values for the tuples. Values must be in the same order.

Creation of table 'BRANCH'

It has the following attributes: branch\_name, branch\_city, assets.

```
CREATE TABLE BRANCH
  (branch_name      varchar2(25) PRIMARY KEY,
   branch_city      varchar2(20) NOT NULL,
   assets           decimal(10, 2)NOT NULL);
```

Creation of table 'ACCOUNT'

It has the following attributes: accno, branch\_name, balance.

```
CREATE TABLE ACCOUNT
  (accno           int                PRIMARY KEY,
   branch_name     varchar2(25) NOT NULL,
   balance         decimal(10, 2)NOT NULL,
   FOREIGN KEY(branch_name) references BRANCH(branch_name));
```

Creation of table 'DEPOSITOR'

It has the following attributes: customer\_name, accno.

```
CREATE TABLE DEPOSITOR
  (customer_name   varchar2(25) NOT NULL,
   accno           int                NOT NULL,
   FOREIGN KEY(accno) references ACCOUNT(accno)
   FOREIGN KEY(customer_name) references
   CUSTOMER(customer_name)
   PRIMARY KEY(customer_name, accno));
```

Creation of table 'CUSTOMER'

It has the following attributes: customer\_name, customer\_street, customer\_city.

```
CREATE TABLE CUSTOMER
  (customer_name   varchar2(25) PRIMARY KEY,
   customer_street  varchar2(20) NOT NULL,
   customer_city    varchar2(20) NOT NULL);
```

Creation of table 'LOAN'

It has the following attributes: loan\_number, branch\_name, amount.

```
CREATE TABLE LOAN
  (loan_number     int                PRIMARY KEY,
   branch_name     varchar2(25) NOT NULL,
   amount          decimal(10, 2)NOT NULL,
   FOREIGN KEY(branch_name) references BRANCH(branch_name));
```

Creation of table 'BORROWER'

It has the following attributes: customer\_name, loan\_number.

**CREATE TABLE BORROWER**

```
(customer_name    varchar2(25) NOT NULL,
 loan_numberint    NOT NULL,
 FOREIGN KEY(customer_name) references
 CUSTOMER(customer_name),
 FOREIGN KEY(loan_number) references LOAN(loan_number).
 PRIMARY KEY(customer_name, loan_number));
```

Insertion of tuples into the table 'BRANCH':

```
INSERT INTO BRANCH VALUES('Citibank', 'Bangalore', 10000.233);
INSERT INTO BRANCH VALUES('HDFC', 'Mumbai', 25675.988);
```

Insertion of tuples into the table 'ACCOUNT':

```
INSERT INTO ACCOUNT VALUES(100, 'Citibank', 8000.233);
INSERT INTO ACCOUNT VALUES(122, 'Citibank', 4556.988);
```

Insertion of tuples into the table 'DEPOSITOR':

```
INSERT INTO DEPOSITOR VALUES('Smith', 100);
INSERT INTO DEPOSITOR VALUES('John', 122);
```

Insertion of tuples into the table 'CUSTOMER':

```
INSERT INTO CUSTOMER VALUES('John', 'Mount street', 'Mumbai');
INSERT INTO CUSTOMER VALUES('Smith', 'Lincoln street', 'Norwok');
```

Insertion of tuples into the table 'LOAN':

```
INSERT INTO LOAN VALUES(901, 'HDFC', 23455.22);
INSERT INTO LOAN VALUES(877, 'Citibank', 7887.09);
```

Insertion of tuples into the table 'BORROWER':

```
INSERT INTO BORROWER VALUES('Smith', 877);
INSERT INTO BORROWER VALUES('John', 901);
```

## QUERIES

**STATEMENT:** Find all the customers who have at least two accounts at the *Main* branch.

**QUERY:**

```
select b_name, cust_name, count (*)
from account, depositor
where accno = actno and b_name = 'main'
group by cust_name, b_name
having count (*) >= 2
```

**EXPLANATION:**

The above query is similar to a SELECT-JOIN-PROJECT sequence of relational algebra operations and such queries are called select-join queries. In the **where** clause, `b_name = 'MAIN'` specifies the selection condition and `accno = actno` is a join condition for the join operation on the two relations `account` and `depositor`. Then the **group by** clause is used to sub-group the tuples based on the grouping attributes `b_name` and `cust_name`. The **having** clause provides a condition `count (*) >= 2` on the groups of tuples. Only the groups that satisfy the condition are retrieved in the result of the query.

**RESULT:**

<u>b_name</u>	<u>cust_name</u>	<u>count (*)</u>
MAIN	TOM	2

The result of the query is as shown above. It selects only the customers who have two or more accounts in the MAIN branch.

**STATEMENT:** Find all the customers who have an account at *all* the branches located in a specific city.

**QUERY:**

```
select accno, cust_name
from account, depositor
where accno=actno and b_name in (select branch_name
                                from branch
                                where branch_city = 'Bangalore')
group by accno, cust_name
```

**EXPLANATION:**

The nested query selects the tuples that satisfy the selection condition `branch_city = 'Bangalore'` from the relation `branch`. In the outer query, a join operation is performed on the relations `account` and `depositor` with the join condition as `accno=actno`. Further the IN operator compares the tuples selected from `account` and `depositor` with the set of union-compatible tuples produced from the nested query for the value `b_name`. The

outer query also uses a group by clause to group the tuples based on the grouping attributes accno, cust\_name.

**RESULT:**

<u>accno</u>	<u>cust_name</u>
456	Eliza

As demonstrated by the result, the above query selects the customers who have account at all the branches located in the city 'Bangalore'.

**STATEMENT:** Demonstrate how you delete all account tuples at every branch located in a specific city.

**QUERY:**

```

delete
from account
where b_name in ( select branch_name
                  from branch
                  where branch_city = 'Sydney')

```

**EXPLANATION:**

The nested query selects the tuples that satisfy the selection condition branch\_city = 'Sydney' from the relation branch. The IN operator compares the subtuple of value b\_name for each tuple in account relation with the tuples produced by the nested query. Finally, the selected tuples are deleted from the account relation.

**RESULT:**

The above query deletes all the account tuples for every branch located in the city 'Sydney'.

## GENERATION OF REPORT

1. Display the names of all the customer who have taken loan, count of the number of loans taken and the total amount taken up each customer.

```

select b.customer_name,count (l.loan_number) as no_of_loans , sum(l.amount) as
total_amount_taken
from borrower b, loan l
where l.loan_number=b.loan_number
groupby b.customer_name;

```

The above query is used to obtain the names of all the customers who have taken loan, the number of loans they have taken and the total amount of money they have taken. This is done by using two tables namely BORROWER and LOAN. The names of all the customer is grouped and the number of loans they have taken is obtained using the count keyword and the total amount of money taken is obtained by using the sum keyword .

#### OUTPUT:

CUST_NAME	NO_OF_LOANS	TOTAL_AMOUNT_TAKEN
aaa	1	3434.3443
ccc	1	889.91
ddd	2	4334.163
eee	1	889.91

2. Find the names of all the customers who have taken the loan in a particular branch.

```

select b.customer_name
from borrower b, loan l
where l.loan_number=b.loan_number and l.branch='gandhi bazar';

```

#### EXPLANATION:

The above query selects the names of all the customer who have taken loans in a particular branch. It performs the action using the two tables LOAN and BORROWER. It compares the loan\_number in both the tables and selects all the common loan\_numbers . From this set it selects those for which the corresponding branch value is BANGALORE. The corresponding customer\_names of the loan\_numbers selected is the required answer.

#### OUTPUT:

```

CUST_NAME
-----
Aaa
Bbb

```

Ccc

3. Update the balance for the customer with a given customer name to 10000

```
update account  
set amount=10000  
where accno in (select accno from depositor where d.customer_name='xyz');
```

**OUTPUT:**

2 rows updated

**EXPLANATION:**

The above query can be performed by using the two tables namely ACCOUNT, and DEPOSITOR . The accno of the depositors whose names are xyz are selected from depositor .For all the selected accno the corresponding amount in the account table is updated to 10000.